ARL-TN-0759 • MAY 2016

US Army Research Laboratory

Resolving the Orientation of Cylinders and Cuboids from Projected Area Measurements

by Richard Saucier

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TN-0759 ● MAY 2016

US Army Research Laboratory

Resolving the Orientation of Cylinders and Cuboids from Projected Area Measurements

by Richard Saucier
Survivability/Lethality Analysis Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
May 2016	Final	November 2015–January 2016

4. TITLE AND SUBTITLE
Resolving the Orientation of Cylinders and Cuboids from Projected Area Measurements

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Richard Saucier

5d. PROJECT NUMBER
AH80

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-SLB-S
Aberdeen Proving Ground, MD 21005-5068

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-0759

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The FATEPEN model predicts the penetration of a mass striking a target plate for a variety of shapes, including cylinders and cuboids—among others. Crucial to the use of the model is a good estimate of not only the mass and velocity but also the impact orientation in terms of pitch, yaw, and roll. Yaw cards and orthogonal X-rays can provide estimates of the impact projected area, but the model makes use of the impact angle, which is defined to be the minimum angle that a face makes with the target plate. This report addresses this issue by calculating the pitch-yaw-roll rotation sequence that will bring about the orientation at impact from orthogonal projected area measurements. It is shown that the impact angle is uniquely determined in the case of cylinders, but that is not the case for cuboids. Furthermore, for cuboids multiple impact angles for the same projected area measurements can lead to significantly different FATEPEN predictions. This leads to the conclusion that cylinders and not cuboids should be used for FATEPEN validation.

15. SUBJECT TERMS
FATEPEN, RCC, RPP, fragment impact, fragment orientation, projected area, orthogonal X-rays, pitch-yaw-roll

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Richard Saucier
Unclassified	Unclassified	Unclassified	UU	44	19b. TELEPHONE NUMBER (Include area code) 410-278-6721

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

List of Listings

Acknowledgments

I would like to thank Timothy Mallory for his interest in this problem, suggestions that improved this report, and the numerous stand-alone FATEPEN runs he conducted to check the results. I would also like to thank John Auten for catching some errors and his technical review of this document. Of course, I accept full responsibility for any errors that may remain.

1. Summary

The FATEPEN model predicts the penetration of a mass striking a target plate for a variety of shapes, including cylinders and cuboids—among others. Crucial to the use of the model is a good estimate of not only the mass and velocity but also the impact orientation in terms of pitch, yaw, and roll. Yaw cards and orthogonal X-rays can provide estimates of the impact projected area, but the model makes use of the impact angle, which is defined to be the minimum angle that a penetrator face makes with the target plate. This report addresses this issue by calculating the pitch-yaw-roll rotation sequence that will bring about the orientation at impact from orthogonal projected area measurements. It is shown that the impact angle is uniquely determined in the case of cylinders, but that is not the case for cuboids. Furthermore, for cuboids multiple impact angles for the same projected area measurements can lead to significantly different FATEPEN predictions. This leads to the conclusion that cylinders and not cuboids should be used for FATEPEN validation.

2. Introduction

The FATEPEN[1] model is a Fortran code for simulating penetration of fragments, long rods, and projectiles into target plates and predicting penetration, perforation, residual mass, and residual velocity. It provides the following shapes: sphere, right circular cylinder (RCC), round-nose cylinder, sharp-nose cylinder, tapered/truncated cylinder, and rectangular parallelepiped (RPP). This report is limited to RCCs, which we will simply call cylinders, and RPPs, which we will call cuboids. Unlike the THOR[2] model, which only requires an impact presented area, the FATEPEN model requires a specific shape and orientation.

The impact presented area, or projected area, is a key parameter that affects the penetration process since the penetration depth scales with the mass per unit presented area for a given striking velocity. But the FATEPEN model goes beyond the presented area and also requires the angle that each flat face of the striking mass makes with the target plate. This means that we need to account for both the presented area and the orientation of the fragment to use FATEPEN properly.

3. Methods, Assumptions, and Procedures

We first define a *standard orientation* of the cylinder or cuboid as the starting orientation and then describe the operational procedure to give it a specific final orientation. This standard orientation will be with the center at the origin of a right-handed Cartesian (x, y, z) coordinate system and the length of the cylinder or cuboid aligned with the z-axis. (The target plate is taken to be parallel to the x-y plane, and the fragment velocity vector is along the negative z axis.) The operational procedure will be in terms pitch, yaw, and roll, where *pitch* is a counterclockwise rotation about the x-axis, *yaw* is a counterclockwise rotation about the y-axis, and *roll* is a counterclockwise rotation about the z-axis.* For convenience, we also define $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, and $\hat{\mathbf{k}}$ to be unit vectors along the x, y, and z axis, respectively, as shown in Fig. 1.

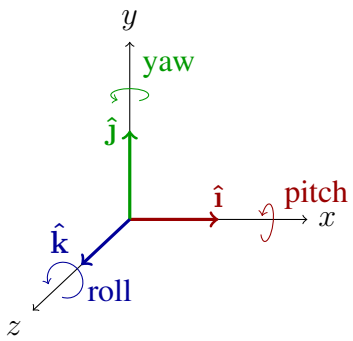

Fig. 1. Description of pitch, yaw, and roll

Pitch, yaw, and roll are then described by the operators $R_{\hat{\mathbf{i}}}(\phi_p)$, $R_{\hat{\mathbf{j}}}(\phi_y)$, and $R_{\hat{\mathbf{k}}}(\phi_r)$, where the unit vector subscript denotes the axis of rotation, and ϕ_p, ϕ_y, and ϕ_r are pitch, yaw, and roll angles, respectively. A *rotation sequence* accounts for the fact that the unit vectors are also transformed by prior rotations. Thus, a pitch-yaw-roll rotation sequence is described by

$$R = R_{\hat{\mathbf{k}}''}(\phi_r) R_{\hat{\mathbf{j}}'}(\phi_y) R_{\hat{\mathbf{i}}}(\phi_p), \tag{1}$$

where the operators are applied from right to left—first pitch, then yaw, then roll—and where $\hat{\mathbf{j}}'$ is the transformed $\hat{\mathbf{j}}$ vector after the pitch rotation, and $\hat{\mathbf{k}}''$ is the doubly transformed $\hat{\mathbf{k}}$ vector after the pitch and yaw rotations. Now we make use of the fact that we generate the same net rotation if we apply the sequence in *reverse* order

*To avoid any confusion, note that FATEPEN uses a pitch-yaw-roll rotation sequence, which we follow in this report, but much of the aerospace industry uses a yaw-pitch-roll rotation sequence.

about the *fixed* axes.* This means that

$$\boxed{R = R_{\hat{\mathbf{k}}''}(\phi_r) R_{\hat{\mathbf{j}}'}(\phi_y) R_{\hat{\mathbf{i}}}(\phi_p) = R_{\hat{\mathbf{i}}}(\phi_p) R_{\hat{\mathbf{j}}}(\phi_y) R_{\hat{\mathbf{k}}}(\phi_r)} \tag{2}$$

and we only have to concern ourselves with rotations about fixed axes. Table 1 lists all $3 \times 3 = 9$ combinations of rotations applied to the unit vectors.

Table 1. Pitch, yaw, and roll rotations of unit vectors

Rotation	$\hat{\mathbf{i}}$	$\hat{\mathbf{j}}$	$\hat{\mathbf{k}}$
Pitch: $R_{\hat{\mathbf{i}}}(\phi_p)$	$\hat{\mathbf{i}}$	$\cos\phi_p\hat{\mathbf{j}} + \sin\phi_p\hat{\mathbf{k}}$	$-\sin\phi_p\hat{\mathbf{j}} + \cos\phi_p\hat{\mathbf{k}}$
Yaw: $R_{\hat{\mathbf{j}}}(\phi_y)$	$\cos\phi_y\hat{\mathbf{i}} - \sin\phi_y\hat{\mathbf{k}}$	$\hat{\mathbf{j}}$	$\sin\phi_y\hat{\mathbf{i}} + \cos\phi_y\hat{\mathbf{k}}$
Roll: $R_{\hat{\mathbf{k}}}(\phi_r)$	$\cos\phi_r\hat{\mathbf{i}} + \sin\phi_r\hat{\mathbf{j}}$	$-\sin\phi_r\hat{\mathbf{i}} + \cos\phi_r\hat{\mathbf{j}}$	$\hat{\mathbf{k}}$

3.1 Orientation of a Cylinder

The cylinder has a length L and diameter D. In standard orientation the center of the cylinder is at the origin and the length is aligned with the z-axis.

3.1.1 Projected Areas from Cylinder Orientation

So let us apply a pitch-yaw-roll rotation sequence to the cylinder in standard orientation and work out the projected areas on the orthogonal y-z, x-z, and x-y planes. The orientation of the cylinder is completely described by its axis of rotation, which is along the unit vector $\hat{\mathbf{k}}$. Let $\hat{\mathbf{u}}$ be the final orientation of the $\hat{\mathbf{k}}$ vector. Then, using Table 1, we get

$$\begin{aligned}
\hat{\mathbf{u}} &= R_{\hat{\mathbf{i}}}(\phi_p) R_{\hat{\mathbf{j}}}(\phi_y) \hat{\mathbf{k}} \\
&= R_{\hat{\mathbf{i}}}(\phi_p)(\sin\phi_y\hat{\mathbf{i}} + \cos\phi_y\hat{\mathbf{k}}) \\
&= \sin\phi_y\hat{\mathbf{i}} + \cos\phi_y(-\sin\phi_p\hat{\mathbf{j}} + \cos\phi_p\hat{\mathbf{k}}).
\end{aligned} \tag{3}$$

It is also convenient to specify $\hat{\mathbf{u}}$ in terms of the direction cosines:

$$\hat{\mathbf{u}} = \cos\phi_1\hat{\mathbf{i}} + \cos\phi_2\hat{\mathbf{j}} + \cos\phi_3\hat{\mathbf{k}}, \tag{4}$$

*See Appendix A for a derivation of this result.

where $\cos\phi_1 = \hat{\mathbf{u}} \cdot \hat{\mathbf{i}}$, $\cos\phi_2 = \hat{\mathbf{u}} \cdot \hat{\mathbf{j}}$, and $\cos\phi_3 = \hat{\mathbf{u}} \cdot \hat{\mathbf{k}}$. Comparing Eqs. 3 and 4 gives the correspondence

$$\begin{aligned} \phi_1 &= \cos^{-1}(\sin\phi_y), \\ \phi_2 &= \cos^{-1}(-\sin\phi_p \cos\phi_y), \\ \phi_3 &= \cos^{-1}(\cos\phi_p \cos\phi_y). \end{aligned} \tag{5}$$

Let L be the length of the cylinder and D be its diameter. Then the projected areas onto the orthogonal planes are

$$\begin{aligned} A_{yz} &= LD|\cos\left(\frac{\pi}{2} - \phi_1\right)| + \frac{\pi}{4}D^2|\cos\phi_1| = LD|\sin\phi_1| + \frac{\pi}{4}D^2|\cos\phi_1| \\ A_{xz} &= LD|\cos\left(\frac{\pi}{2} - \phi_2\right)| + \frac{\pi}{4}D^2|\cos\phi_2| = LD|\sin\phi_2| + \frac{\pi}{4}D^2|\cos\phi_2|\,. \\ A_{xy} &= LD|\cos\left(\frac{\pi}{2} - \phi_3\right)| + \frac{\pi}{4}D^2|\cos\phi_3| = LD|\sin\phi_3| + \frac{\pi}{4}D^2|\cos\phi_3| \end{aligned} \tag{6}$$

Therefore, given the pitch and yaw of the cylinder, Eqs. 5 and 6 give us the projected areas.

3.1.2 Cylinder Orientation from Projected Areas

Next, we work out the inverse problem of determining the pitch and yaw from the measured projected areas. Thus, given the length L, diameter D, and the projected areas of an RCC onto the orthogonal planes, we want to solve for the orientation of the RCC. More specifically, we seek the FATEPEN pitch and yaw rotation sequence that will bring about this orientation.* It is convenient in what follows to specify the final orientation in terms of the polar angle, θ, measured from the z-axis and ϕ, the azimuthal angle measured from the x-axis in the x-y plane:

$$\hat{\mathbf{u}} = \sin\theta\cos\phi\hat{\mathbf{i}} + \sin\theta\sin\phi\hat{\mathbf{j}} + \cos\theta\hat{\mathbf{k}}. \tag{7}$$

There is no loss of generality if we restrict $\hat{\mathbf{u}}$ to lie in the first octant of the unit sphere, so that

$$0 \le \theta \le \pi/2 \quad \text{and} \quad 0 \le \phi \le \pi/2. \tag{8}$$

*Ordinarily we would need pitch, yaw, and roll to completely specify orientation, but placing the axis of the RCC along the z (roll) axis eliminates the need to consider roll.

Comparing Eqs. 4 and 7 gives

$$\theta = \phi_3 \quad \text{and} \quad \phi = \tan^{-1}\left(\frac{\cos\phi_2}{\cos\phi_1}\right). \tag{9}$$

All the formulas in Eq. 6 are of the form

$$a\sin\alpha + b\cos\alpha = A_p, \tag{10}$$

with $\alpha = \phi_1$, ϕ_2, or ϕ_3, $a = LD$, $b = \pi D^2/4$, and A_p the projected area, A_{yz}, A_{xz}, or A_{xy}. We can solve this for the angle α by first introducing another angle β, defined by

$$\beta \equiv \tan^{-1}(b/a), \tag{11}$$

which gives $a = \sqrt{a^2+b^2}\cos\beta$ and $b = \sqrt{a^2+b^2}\sin\beta$, and then Eq. 10 becomes

$$\sqrt{a^2+b^2}\cos\beta\sin\alpha + \sqrt{a^2+b^2}\sin\beta\cos\alpha = \sqrt{a^2+b^2}\sin(\alpha+\beta) = A_p. \tag{12}$$

Now it is easy to show that $\sqrt{a^2+b^2} = A_{\text{max}}$, the maximum presented area of the RCC, so that $0 \le A_p/\sqrt{a^2+b^2} \le 1$, and therefore

$$\sin(\alpha+\beta) = \frac{A_p}{\sqrt{a^2+b^2}}. \tag{13}$$

Two possible solutions to this equation are

$$\alpha = \sin^{-1}\left(\frac{A_p}{\sqrt{a^2+b^2}}\right) - \tan^{-1}\left(\frac{b}{a}\right) \quad \text{or}$$

$$\alpha = \pi - \tan^{-1}\left(\frac{b}{a}\right) - \sin^{-1}\left(\frac{A_p}{\sqrt{a^2+b^2}}\right). \tag{14}$$

In practice we try both, so this gives $2^3 = 8$ possible combinations for ϕ_1, ϕ_2, and ϕ_3. But by imposing the constraint that

$$\cos^2\phi_1 + \cos^2\phi_2 + \cos^2\phi_3 = 1, \tag{15}$$

we will find that only one combination will work. Once ϕ_1, ϕ_2, and ϕ_3 have thus been found, we return to Eq. 9 to get θ and ϕ.

Next we need a rotation that will align $\hat{\mathbf{k}}$ with $\hat{\mathbf{u}}$. It is easy to check that $R_{\hat{\mathbf{e}}}(\theta)$, where

$$\hat{\mathbf{e}} = \frac{\hat{\mathbf{k}} \times \hat{\mathbf{u}}}{|\hat{\mathbf{k}} \times \hat{\mathbf{u}}|} = -\sin\phi\hat{\mathbf{i}} + \cos\phi\hat{\mathbf{j}}, \tag{16}$$

is this rotation. That is, $R_{\hat{\mathbf{e}}}(\theta)\hat{\mathbf{k}} = \hat{\mathbf{u}}$. The quaternion representation of this rotation is

$$q_{\hat{\mathbf{e}}}(\theta) = \cos(\theta/2) + \hat{\mathbf{e}}\sin(\theta/2) = \cos(\theta/2) + (-\sin\phi\hat{\mathbf{i}} + \cos\phi\hat{\mathbf{j}})\sin(\theta/2). \tag{17}$$

Finally, this quaternion can be factored into a pitch-yaw-roll rotation sequence.[3] The prescription is as follows:

1. Set $p_0 = \cos(\theta/2)$, $p_1 = -\sin\phi\sin(\theta/2)$, $p_2 = \cos\phi\sin(\theta/2)$, $p_3 = 0$.
2. Set $A = p_1p_2 - p_0p_3$, $B = p_1^2 - p_3^2$, $D = p_0^2 - p_2^2$.
3. Then $\phi_r = \tan^{-1}[-2A/(B+D)]$.
4. Set $c_0 = \cos(\phi_r/2)$ and $c_3 = \sin(\phi_r/2)$.
5. Set $q_0 = p_0c_0 + p_3c_3$, $q_1 = p_1c_0 - p_2c_3$, $q_2 = p_2c_0 + p_1c_3$, $q_3 = p_3c_0 - p_0c_3$.
6. Then $\phi_p = 2\tan^{-1}(q_1/q_0)$ and $\phi_y = 2\tan^{-1}(q_2/q_0)$.

Thus, we accomplish what we set out to do: obtain the pitch and yaw rotation sequence that will give the desired projected areas. For a cylinder there are actually 4 orientations that give the same projected areas, which are (ϕ_p, ϕ_y), $(\phi_p, -\phi_y)$, $(-\phi_p, \phi_y)$, and $(-\phi_p, -\phi_y)$.

3.2 Orientation of a Cuboid

The cuboid, or RPP, has a length L, width W, and thickness T, where $L \geq W \geq T$, and is initially oriented so that the length is along the z-axis, the width is along the x-axis and the thickness is along the y-axis. Thus, the initial areas along each of the axes are $A_x = LT$, $A_y = LW$, and $A_z = WT$. We take $\hat{\mathbf{i}}$ to be a unit vector along the x-axis, $\hat{\mathbf{j}}$ to be a unit vector along the y-axis, and $\hat{\mathbf{k}}$ to be a unit vector along the z-axis. Let

$$\hat{\mathbf{u}} = x\hat{\mathbf{i}} + y\hat{\mathbf{j}} + z\hat{\mathbf{k}} \quad \text{with} \quad x^2 + y^2 + z^2 = 1 \tag{18}$$

be a unit vector on the unit sphere. Then the projected area of the cuboid orthogonal to $\hat{\mathbf{u}}$ is

$$A_p(x, y, z) = (A_x\hat{\mathbf{i}} + A_y\hat{\mathbf{j}} + A_z\hat{\mathbf{k}}) \cdot \hat{\mathbf{u}} = A_x x + A_y y + A_z z. \tag{19}$$

As we vary the direction of $\hat{\mathbf{u}}$, the rate of change of the projected area is given by

$$\begin{aligned} D_{\hat{\mathbf{u}}} A_p &= \boldsymbol{\nabla} A_p(x, y, z) \cdot \hat{\mathbf{u}} \\ &= ||\boldsymbol{\nabla} A_p(x, y, z)||\ ||\hat{\mathbf{u}}|| \cos\theta \\ &= ||\boldsymbol{\nabla} A_p(x, y, z)|| \cos\theta, \end{aligned} \tag{20}$$

where θ is the angle between the gradient and the unit vector. From this expression, it is clear that the area is maximized when $\cos\theta$ is a maximum, which occurs when $\theta = 0$. The gradient along this direction is

$$\boldsymbol{\nabla} A_p(x, y, z) = \frac{\partial A_p}{\partial x}\hat{\mathbf{i}} + \frac{\partial A_p}{\partial y}\hat{\mathbf{j}} + \frac{\partial A_p}{\partial z}\hat{\mathbf{k}} = A_x\hat{\mathbf{i}} + A_y\hat{\mathbf{j}} + A_z\hat{\mathbf{k}}, \tag{21}$$

and therefore the magnitude of the maximum projected area is

$$A_{\max} = ||\boldsymbol{\nabla} A_p(x, y, z)|| = \sqrt{A_x^2 + A_y^2 + A_z^2}\,, \tag{22}$$

and the view direction for this maximum is given by the unit vector

$$\hat{\mathbf{u}}_{\max} = \frac{A_x\hat{\mathbf{i}} + A_y\hat{\mathbf{j}} + A_z\hat{\mathbf{k}}}{A_{\max}}. \tag{23}$$

By construction, the minimum projected area is initially oriented along the z-axis, $A_{\min} = A_z$, and is realized along the unit vector $\hat{\mathbf{u}}_{\min} = \hat{\mathbf{k}}$.

3.2.1 Projected Areas from Cuboid Orientation

Now let us work out the projected areas from a given pitch-yaw-roll rotation sequence, R. Again, making use of Eq. 2 and applying the rotations from Table 1 successively gives

$$\begin{aligned} R_{\hat{\mathbf{j}}}(\phi_y) R_{\hat{\mathbf{k}}}(\phi_r) A_x\hat{\mathbf{i}} &= A_x \cos\phi_r(\cos\phi_y\hat{\mathbf{i}} - \sin\phi_y\hat{\mathbf{k}}) + A_x \sin\phi_r\hat{\mathbf{j}}, \\ R_{\hat{\mathbf{j}}}(\phi_y) R_{\hat{\mathbf{k}}}(\phi_r) A_y\hat{\mathbf{j}} &= -A_y \sin\phi_r(\cos\phi_y\hat{\mathbf{i}} - \sin\phi_y\hat{\mathbf{k}}) + A_y \cos\phi_r\hat{\mathbf{j}}, \\ R_{\hat{\mathbf{j}}}(\phi_y) R_{\hat{\mathbf{k}}}(\phi_r) A_z\hat{\mathbf{k}} &= A_z(\sin\phi_y\hat{\mathbf{i}} + \cos\phi_y\hat{\mathbf{k}}). \end{aligned} \tag{24}$$

Collecting terms we get

$$\begin{aligned}
R_{\hat{\mathbf{j}}}(\phi_y)R_{\hat{\mathbf{k}}}(\phi_r)A_x\hat{\mathbf{i}} &= A_x\cos\phi_r\cos\phi_y\hat{\mathbf{i}} + A_x\sin\phi_r\hat{\mathbf{j}} - A_x\cos\phi_r\sin\phi_y\hat{\mathbf{k}},\\
R_{\hat{\mathbf{j}}}(\phi_y)R_{\hat{\mathbf{k}}}(\phi_r)A_y\hat{\mathbf{j}} &= -A_y\sin\phi_r\cos\phi_y\hat{\mathbf{i}} + A_y\cos\phi_r\hat{\mathbf{j}} + A_y\sin\phi_r\sin\phi_y\hat{\mathbf{k}},\\
R_{\hat{\mathbf{j}}}(\phi_y)R_{\hat{\mathbf{k}}}(\phi_r)A_z\hat{\mathbf{k}} &= A_z\sin\phi_y\hat{\mathbf{i}} + A_z\cos\phi_y\hat{\mathbf{k}}.
\end{aligned} \tag{25}$$

Finally, applying pitch, we get

$$\begin{aligned}
RA_x\hat{\mathbf{i}} =& + A_x\cos\phi_r\cos\phi_y\hat{\mathbf{i}}\\
&+ A_x\sin\phi_r(\cos\phi_p\hat{\mathbf{j}} + \sin\phi_p\hat{\mathbf{k}})\\
&- A_x\cos\phi_r\sin\phi_y(-\sin\phi_p\hat{\mathbf{j}} + \cos\phi_p\hat{\mathbf{k}}),\\
RA_y\hat{\mathbf{j}} =& - A_y\sin\phi_r\cos\phi_y\hat{\mathbf{i}}\\
&+ A_y\cos\phi_r(\cos\phi_p\hat{\mathbf{j}} + \sin\phi_p\hat{\mathbf{k}})\\
&+ A_y\sin\phi_r\sin\phi_y(-\sin\phi_p\hat{\mathbf{j}} + \cos\phi_p\hat{\mathbf{k}}),\\
RA_z\hat{\mathbf{k}} =& + A_z\sin\phi_y\hat{\mathbf{i}}\\
&+ A_z\cos\phi_y(-\sin\phi_p\hat{\mathbf{j}} + \cos\phi_p\hat{\mathbf{k}}).
\end{aligned} \tag{26}$$

and collecting terms,

$$\begin{aligned}
RA_x\hat{\mathbf{i}} =& + A_x\cos\phi_r\cos\phi_y\hat{\mathbf{i}}\\
&+ A_x(\sin\phi_r\cos\phi_p + \cos\phi_r\sin\phi_y\sin\phi_p)\hat{\mathbf{j}}\\
&+ A_x(\sin\phi_r\sin\phi_p - \cos\phi_r\sin\phi_y\cos\phi_p)\hat{\mathbf{k}},
\end{aligned} \tag{27}$$

$$\begin{aligned}
RA_y\hat{\mathbf{j}} =& - A_y\sin\phi_r\cos\phi_y\hat{\mathbf{i}}\\
&+ A_y(\cos\phi_r\cos\phi_p - \sin\phi_r\sin\phi_y\sin\phi_p)\hat{\mathbf{j}}\\
&+ A_y(\cos\phi_r\sin\phi_p + \sin\phi_r\sin\phi_y\cos\phi_p)\hat{\mathbf{k}},
\end{aligned} \tag{28}$$

$$RA_z\hat{\mathbf{k}} = + A_z\sin\phi_y\hat{\mathbf{i}} - A_z\cos\phi_y\sin\phi_p\hat{\mathbf{j}} + A_z\cos\phi_y\cos\phi_p\hat{\mathbf{k}}. \tag{29}$$

Now let A_{xy} be the projection on the x-y plane, A_{xz} be the projection on the x-z plane, and A_{yz} be the projection on the y-z plane. Then we have

$$A_{xy} = [R(A_x\hat{\mathbf{i}} + A_y\hat{\mathbf{j}} + A_z\hat{\mathbf{k}})]\cdot\hat{\mathbf{k}}, \tag{30}$$

$$A_{xz} = [R(A_x\hat{\mathbf{i}} + A_y\hat{\mathbf{j}} + A_z\hat{\mathbf{k}})]\cdot\hat{\mathbf{j}}, \tag{31}$$

$$A_{yz} = [R(A_x\hat{\mathbf{i}} + A_y\hat{\mathbf{j}} + A_z\hat{\mathbf{k}})]\cdot\hat{\mathbf{i}}. \tag{32}$$

Three is the maximum number of sides that can project unto a plane for a cuboid, so we can include all the contributions by using the absolute value. Thus, we get the projected areas

$$\begin{aligned} A_{xy} =& A_x|\sin\phi_r \sin\phi_p - \cos\phi_r \sin\phi_y \cos\phi_p| + \\ & A_y|\cos\phi_r \sin\phi_p + \sin\phi_r \sin\phi_y \cos\phi_p| + \\ & A_z|\cos\phi_y \cos\phi_p|, \end{aligned} \tag{33}$$

$$\begin{aligned} A_{xz} =& A_x|\sin\phi_r \cos\phi_p + \cos\phi_r \sin\phi_y \sin\phi_p| + \\ & A_y|\cos\phi_r \cos\phi_p - \sin\phi_r \sin\phi_y \sin\phi_p| + \\ & A_z|\cos\phi_y \sin\phi_p|, \end{aligned} \tag{34}$$

$$A_{yz} = A_x|\cos\phi_r \cos\phi_y| + A_y|\sin\phi_r \cos\phi_y| + A_z|\sin\phi_y|. \tag{35}$$

3.2.2 Cuboid Orientation from Projected Areas

Next, given measured values for A_{xy}, A_{xz}, and A_{yz}, we want to solve Eqs. 33–35 for ϕ_p, ϕ_y, and ϕ_r. This is a system of 3 simultaneous nonlinear equations, which can be solved numerically by Newton's method in 3 dimensions. Define

$$\begin{aligned} f(\phi_p, \phi_y, \phi_r) \equiv& A_x|\sin\phi_r \sin\phi_p - \cos\phi_r \sin\phi_y \cos\phi_p| + \\ & A_y|\cos\phi_r \sin\phi_p + \sin\phi_r \sin\phi_y \cos\phi_p| + \\ & A_z|\cos\phi_y \cos\phi_p| - A_{xy}, \end{aligned} \tag{36}$$

$$\begin{aligned} g(\phi_p, \phi_y, \phi_r) \equiv& A_x|\sin\phi_r \cos\phi_p + \cos\phi_r \sin\phi_y \sin\phi_p| + \\ & A_y|\cos\phi_r \cos\phi_p - \sin\phi_r \sin\phi_y \sin\phi_p| + \\ & A_z|\cos\phi_y \sin\phi_p| - A_{xz}, \end{aligned} \tag{37}$$

$$h(\phi_p, \phi_y, \phi_r) \equiv A_x|\cos\phi_r \cos\phi_y| + A_y|\sin\phi_r \cos\phi_y| + A_z|\sin\phi_y| - A_{yz}. \tag{38}$$

Then the problem is to find the roots of f, g, and h. Let A be the matrix of partial derivatives

$$A = \begin{bmatrix} f_{\phi_p}(\phi_p, \phi_y, \phi_r) & f_{\phi_y}(\phi_p, \phi_y, \phi_r) & f_{\phi_r}(\phi_p, \phi_y, \phi_r) \\ g_{\phi_p}(\phi_p, \phi_y, \phi_r) & g_{\phi_y}(\phi_p, \phi_y, \phi_r) & g_{\phi_r}(\phi_p, \phi_y, \phi_r) \\ h_{\phi_p}(\phi_p, \phi_y, \phi_r) & h_{\phi_y}(\phi_p, \phi_y, \phi_r) & h_{\phi_r}(\phi_p, \phi_y, \phi_r) \end{bmatrix} \tag{39}$$

where

$$f_{\phi_p}(\phi_p, \phi_y, \phi_r) = \frac{\partial f(\phi_p, \phi_y, \phi_r)}{\partial \phi_p} \tag{40}$$

and so on for the other partial derivatives. For this we need the derivative of the absolute value:

$$\frac{d}{dx}|u| = \frac{d}{dx}\sqrt{u^2} = \frac{1}{2}(u^2)^{-1/2}2uu' = \frac{uu'}{\sqrt{u^2}} = \frac{uu'}{|u|} = \operatorname{sgn}(u)\,u', \tag{41}$$

where $\operatorname{sgn}(x)$ is the sign function:

$$\operatorname{sgn}(x) = \begin{cases} +1 & \text{if } x > 0 \\ 0 & \text{if } x = 0\,. \\ -1 & \text{if } x < 0 \end{cases} \tag{42}$$

Then from Eqs. 36–38 we get

$$\begin{aligned} f_{\phi_p}(\phi_p, \phi_y, \phi_r) = {} & +A_x \operatorname{sgn}(\sin\phi_r \sin\phi_p - \cos\phi_r \sin\phi_y \cos\phi_p) \\ & (\sin\phi_r \cos\phi_p + \cos\phi_r \sin\phi_y \sin\phi_p) \\ & +A_y \operatorname{sgn}(\cos\phi_r \sin\phi_p + \sin\phi_r \sin\phi_y \cos\phi_p) \\ & (\cos\phi_r \cos\phi_p - \sin\phi_r \sin\phi_y \sin\phi_p) \\ & +A_z \operatorname{sgn}(\cos\phi_y \cos\phi_p)(-\cos\phi_y \sin\phi_p), \end{aligned} \tag{43}$$

$$\begin{aligned} f_{\phi_y}(\phi_p, \phi_y, \phi_r) = {} & +A_x \operatorname{sgn}(\sin\phi_r \sin\phi_p - \cos\phi_r \sin\phi_y \cos\phi_p) \\ & (-\cos\phi_r \cos\phi_y \sin\phi_p) \\ & +A_y \operatorname{sgn}(\cos\phi_r \sin\phi_p + \sin\phi_r \sin\phi_y \cos\phi_p) \\ & (\sin\phi_r \cos\phi_y \cos\phi_p) \\ & +A_z \operatorname{sgn}(\cos\phi_y \cos\phi_p)(-\sin\phi_y \cos\phi_p), \end{aligned} \tag{44}$$

$$\begin{aligned} f_{\phi_r}(\phi_p, \phi_y, \phi_r) = {} & +A_x \operatorname{sgn}(\sin\phi_r \sin\phi_p - \cos\phi_r \sin\phi_y \cos\phi_p) \\ & (\cos\phi_r \sin\phi_p + \sin\phi_r \sin\phi_y \cos\phi_p) \\ & +A_y \operatorname{sgn}(\cos\phi_r \sin\phi_p + \sin\phi_r \sin\phi_y \cos\phi_p) \\ & (-\sin\phi_r \sin\phi_p + \cos\phi_r \sin\phi_y \cos\phi_p), \end{aligned} \tag{45}$$

$$
\begin{aligned}
g_{\phi_p}(\phi_p,\phi_y,\phi_r) = &+A_x\,\mathrm{sgn}(\sin\phi_r\cos\phi_p+\cos\phi_r\sin\phi_y\sin\phi_p)\\
&\quad(-\sin\phi_r\sin\phi_p+\cos\phi_r\sin\phi_y\cos\phi_p)\\
&+A_y\,\mathrm{sgn}(\cos\phi_r\cos\phi_p-\sin\phi_r\sin\phi_y\sin\phi_p)\\
&\quad(-\cos\phi_r\sin\phi_p-\sin\phi_r\sin\phi_y\cos\phi_p)\\
&+A_z\,\mathrm{sgn}(\cos\phi_y\sin\phi_p)(\cos\phi_y\cos\phi_p), \qquad (46)\\
g_{\phi_y}(\phi_p,\phi_y,\phi_r) = &+A_x\,\mathrm{sgn}(\sin\phi_r\cos\phi_p+\cos\phi_r\sin\phi_y\sin\phi_p)\\
&\quad(\cos\phi_r\cos\phi_y\sin\phi_p)\\
&+A_y\,\mathrm{sgn}(\cos\phi_r\cos\phi_p-\sin\phi_r\sin\phi_y\sin\phi_p)\\
&\quad(-\sin\phi_r\cos\phi_y\sin\phi_p)\\
&+A_z\,\mathrm{sgn}(\cos\phi_y\sin\phi_p)(-\sin\phi_y\sin\phi_p), \qquad (47)\\
g_{\phi_r}(\phi_p,\phi_y,\phi_r) = &+A_x\,\mathrm{sgn}(\sin\phi_r\cos\phi_p+\cos\phi_r\sin\phi_y\sin\phi_p)\\
&\quad(\cos\phi_r\cos\phi_p-\sin\phi_r\sin\phi_y\sin\phi_p)\\
&+A_y\,\mathrm{sgn}(\cos\phi_r\cos\phi_p-\sin\phi_r\sin\phi_y\sin\phi_p)\\
&\quad(-\sin\phi_r\cos\phi_p-\cos\phi_r\sin\phi_y\sin\phi_p), \qquad (48)\\
h_{\phi_p}(\phi_p,\phi_y,\phi_r) = &\quad 0, \qquad (49)\\
h_{\phi_y}(\phi_p,\phi_y,\phi_r) = &+A_x\,\mathrm{sgn}(\cos\phi_r\cos\phi_y)(-\cos\phi_r\sin\phi_y)+\\
&+A_y\,\mathrm{sgn}(\sin\phi_r\cos\phi_y)(-\sin\phi_r\sin\phi_y)+\\
&+A_z\,\mathrm{sgn}(\sin\phi_y)(\cos\phi_y), \qquad (50)\\
h_{\phi_r}(\phi_p,\phi_y,\phi_r) = &+A_x\,\mathrm{sgn}(\cos\phi_r\cos\phi_y)(-\sin\phi_r\cos\phi_y)\\
&+A_y\,\mathrm{sgn}(\sin\phi_r\cos\phi_y)(\cos\phi_r\cos\phi_y). \qquad (51)
\end{aligned}
$$

Newton's method for solving this 3-dimensional (3D) problem numerically for pitch, yaw, and roll is expressed by the matrix iteration equation

$$
\begin{bmatrix}\delta\phi_p\\ \delta\phi_y\\ \delta\phi_r\end{bmatrix} \equiv \begin{bmatrix}\phi_{p,n+1}-\phi_{p,n}\\ \phi_{y,n+1}-\phi_{y,n}\\ \phi_{r,n+1}-\phi_{r,n}\end{bmatrix} = A^{-1}\begin{bmatrix}f(\phi_p,\phi_y,\phi_r)\\ g(\phi_p,\phi_y,\phi_r)\\ h(\phi_p,\phi_y,\phi_r)\end{bmatrix}, \qquad (52)
$$

where the matrix A is given by Eq. 39. We pick a starting orientation and continue iterating until either the δ's fall below a preset tolerance value ϵ or we exceed a maximum number of iterations—in which case we try another starting orientation.

If we write this coefficient matrix (Eq. 39) as

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \tag{53}$$

then the inverse is given by

$$A^{-1} = \frac{1}{\det A} \begin{bmatrix} a_{22}a_{33} - a_{23}a_{32} & a_{13}a_{32} - a_{12}a_{33} & a_{12}a_{23} - a_{13}a_{22} \\ a_{23}a_{31} - a_{21}a_{33} & a_{11}a_{33} - a_{13}a_{31} & a_{13}a_{21} - a_{11}a_{23} \\ a_{21}a_{32} - a_{22}a_{31} & a_{12}a_{31} - a_{11}a_{32} & a_{11}a_{22} - a_{12}a_{21} \end{bmatrix}. \tag{54}$$

Thus, we now have all the ingredients for implementing the proposed solution with the program in Listing 1.

Listing 1. orient.cpp

```
// orient.cpp: Given the dimensions of an RPP and projected areas onto three orthogonal planes,
//             finds a pitch-yaw-roll rotation sequence that will orient the RPP for these projected areas.
//             The orientation is not unique and will depend upon the initial starting point.
//             Method of solution makes use of Newton's method in three dimensions.
// R. Saucier, October 2015

#include <iostream>
#include <cstdlib>
#include <cmath>
#include <iomanip>
#include <chrono>
#include <random>

inline double sgn( double x ) {

        if ( x > 0. ) return +1.;
   else if ( x < 0. ) return -1.;
   else               return 0.;
}

int main( int argc, char* argv[] ) {

   const double D2R = M_PI / 180.;   // to convert deg to rad
   const double R2D = 180. / M_PI;   // to convert rad to deg
   const double TOL = 1.e-9;         // convergence criterion
   const int    N   = 100;           // max number of iterations

   double L = 3.;   // length
   double W = 2.;   // width
   double T = 1.;   // thickness

   double Ax = L * T;   // area of side (initially along i)
   double Ay = L * W;   // area of top (initially along j)
   double Az = W * T;   // area of front (initially along k)

   double A_xy = 4.5;   // projected area on x-y plane
   double A_xz = 6.5;   // projected area on x-z plane
   double A_yz = 3.5;   // projected area on y-z plane

   double p, y, r, cp, sp, cy, sy, cr, sr, f, g, h;
   double e1, e2, e3, e4, e5, e6, e7, e8, e9;
   double s1, s2, s3, s4, s5, s6, s7, s8, s9;
   double fp, fy, fr, gp, gy, gr, hp, hy, hr;
   double a11, a12, a13, a21, a22, a23, a31, a32, a33, det;
   double b11, b12, b13, b21, b22, b23, b31, b32, b33;
   double del_p, del_y, del_r;

   // initial estimate can't be zero
   unsigned int seed = std::chrono::high_resolution_clock::now().time_since_epoch().count();
   std::mt19937 rng( seed );                                    // Mersenne Twister engine
```

```
    std::uniform_real_distribution<double> uniform( 0., M_PI );   // uniform distribution
    p = uniform( rng );
    y = uniform( rng );
    r = uniform( rng );

    if ( argc == 4 ) { // optionally specify initial pitch, yaw, roll on commandline

       p = atof( argv[1] ) * D2R;
       y = atof( argv[2] ) * D2R;
       r = atof( argv[3] ) * D2R;
    }

    for ( int i = 0; i < N; i++ ) {

       cp = cos( p ); sp = sin( p );
       cy = cos( y ); sy = sin( y );
       cr = cos( r ); sr = sin( r );

       e1 = sr * sp - cr * sy * cp;
       e2 = cr * sp + sr * sy * cp;
       e3 = cy * cp;
       e4 = sr * cp + cr * sy * sp;
       e5 = cr * cp - sr * sy * sp;
       e6 = cy * sp;
       e7 = cr * cy;
       e8 = sr * cy;
       e9 = sy;

       s1 = sgn( e1 );
       s2 = sgn( e2 );
       s3 = sgn( e3 );
       s4 = sgn( e4 );
       s5 = sgn( e5 );
       s6 = sgn( e6 );
       s7 = sgn( e7 );
       s8 = sgn( e8 );
       s9 = sgn( e9 );

       f = Ax * fabs( e1 ) + Ay * fabs( e2 ) + Az * fabs( e3 ) - A_xy;
       g = Ax * fabs( e4 ) + Ay * fabs( e5 ) + Az * fabs( e6 ) - A_xz;
       h = Ax * fabs( e7 ) + Ay * fabs( e8 ) + Az * fabs( e9 ) - A_yz;

       fp = Ax * s1 * ( sr * cp + cr * sy * sp ) + Ay * s2 * ( cr * cp - sr * sy * sp ) + Az * s3 * ( -cy * sp );
       fy = Ax * s1 * ( -cr * cy * sp ) + Ay * s2 * ( sr * cy * cp ) + Az * s3 * ( -sy * cp );
       fr = Ax * s1 * ( cr * sp + sr * sy * cp ) + Ay * s2 * ( -sr * sp + cr * sy * cp );

       gp = Ax * s4 * ( -sr * sp + cr * sy * cp ) + Ay * s5 * ( -cr * sp - sr * sy * cp ) + Az * s6 * ( cy * cp );
       gy = Ax * s4 * ( cr * cy * sp ) + Ay * s5 * ( -sr * cy * sp ) + Az * s6 * ( -sy * sp );
       gr = Ax * s4 * ( cr * cp - sr * sy * sp ) + Ay * s5 * ( -sr * cp - cr * sy * sp );

       hp = 0.;
       hy = Ax * s7 * ( -cr * sy ) + Ay * s8 * ( -sr * sy ) + Az * s9 * ( cy );
       hr = Ax * s7 * ( -sr * cy ) + Ay * s8 * ( cr * cy );

       a11 = fp; a12 = fy; a13 = fr;
       a21 = gp; a22 = gy; a23 = gr;
       a31 = hp; a32 = hy; a33 = hr;
       det = a11 * ( a22 * a33 - a23 * a32 ) + a12 * ( a23 * a31 - a21 * a33 ) + a13 * ( a21 * a32 - a22 * a31 );
       if ( det == 0 ) {

          std::cerr << "bad initial orientation: " << p * R2D << "\t" << y * R2D << "\t" << r * R2D << std::endl
                    << "program " << argv[0] << " stopped" << std::endl;
          exit( EXIT_FAILURE );
       }

       b11 = ( a22 * a33 - a23 * a32 ) / det;
       b12 = ( a13 * a32 - a12 * a33 ) / det;
       b13 = ( a12 * a23 - a13 * a22 ) / det;
       b21 = ( a23 * a31 - a21 * a33 ) / det;
       b22 = ( a11 * a33 - a13 * a31 ) / det;
       b23 = ( a13 * a21 - a11 * a23 ) / det;
       b31 = ( a21 * a32 - a22 * a31 ) / det;
       b32 = ( a12 * a31 - a11 * a32 ) / det;
       b33 = ( a11 * a22 - a12 * a21 ) / det;

       del_p = b11 * f + b12 * g + b13 * h;
       del_y = b21 * f + b22 * g + b23 * h;
       del_r = b31 * f + b32 * g + b33 * h;

       p -= del_p;
       y -= del_y;
       r -= del_r;

       if ( fabs( del_p ) < TOL && fabs( del_y ) < TOL && fabs( del_r ) < TOL ) break;

```

```
136         if ( i == N-1 ) {
137 
138             std::cerr << "failed to converge after " << N << " iterations: f = " << f << " g = " << g << " h = " << h << std
                    ::endl
139                     << "try another initial orientation" << std::endl
140                     << "program " << argv[0] << " stopped" << std::endl;
141             exit( EXIT_FAILURE );
142         }
143     }
144     std::cout << std::setprecision(9) << std::fixed;
145     std::cout  << "pitch = " << fmod( p * R2D, 360. ) << std::endl
146                << "yaw   = " << fmod( y * R2D, 360. ) << std::endl
147                << "roll  = " << fmod( r * R2D, 360. ) << std::endl;
148 
149     return EXIT_SUCCESS;
150 }
```

The program may be compiled and run with the following commands:

```
c++ -O2 -Wall -std=c++11 -o orient orient.cpp -lm
./orient
```

4. Results and Discussion

We now have

- a method for computing the projected areas on orthogonal planes, given the cylinder or cuboid orientation, and
- a method for determining the pitch-yaw-roll rotation sequence to bring about the orientation, given the projected area measurements.

This means that we have a way to check the results. We will show that projected area measurements lead to a unique orientation for a cylinder, but that is not the case for a cuboid.

4.1 Cylinder Orientation Sample Case

Consider an RCC with $L = 1$, $D = 1$, $\phi_p = 30°$, and $\phi_y = 15°$. The projected areas are found to be $A_{yz} = 1.16920$, $A_{xz} = 1.25496$, and $A_{xy} = 1.20494$. Using these values for the projected areas, the program in Listing 2 computes $\phi_p = \pm 30°$ and $\phi_y = \pm 15°$, with the 4 possible orientations displayed in Fig. 2.

Listing 2. rcc.cpp

```
// rcc.cpp: compute pitch and yaw from area projections for an RCC (sample case)
// R. Saucier, June 2006 (Revised October 2015)

#include "Rotation.h"
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <cassert>
#include <iomanip>
using namespace std;

double angle1( double a, double b, double c ) { return asin( c / sqrt( a * a + b * b ) ) - atan( b / a ); }
double angle2( double a, double b, double c ) { return M_PI - asin( c / sqrt( a * a + b * b ) ) - atan( b / a ); }

int main( void ) {
```

```
16
17      const double L = 1., D = 1.;
18      const double A     = L * D;
19      const double B     = 0.25 * M_PI * D * D;
20      const double A_MIN = A < B ? A : B;
21      const double A_MAX = sqrt( A * A + B * B );
22
23      cout << setprecision(6) << fixed;
24      cout << "L     = " << L << endl;
25      cout << "D     = " << D << endl;
26      cout << "A_MIN = " << A_MIN << endl;
27      cout << "A_MAX = " << A_MAX << endl;
28
29      double A_xy = 1.20494;   // projected area on x-y plane
30      double A_xz = 1.25496;   // projected area on x-z plane
31      double A_yz = 1.16920;   // projected area on y-z plane
32
33      cout << "A_xy = " << A_xy << endl;
34      cout << "A_xz = " << A_xz << endl;
35      cout << "A_yz = " << A_yz << endl;
36
37      assert( A_MIN <= A_xy && A_xy <= A_MAX );
38      assert( A_MIN <= A_yz && A_yz <= A_MAX );
39      assert( A_MIN <= A_xz && A_xz <= A_MAX );
40
41      double phi_x[ 2 ], phi_y[ 2 ], phi_z[ 2 ];
42      phi_x[0] = angle1( A, B, A_yz );
43      phi_x[1] = angle2( A, B, A_yz );
44
45      phi_y[0] = angle1( A, B, A_xz );
46      phi_y[1] = angle2( A, B, A_xz );
47
48      phi_z[0] = angle1( A, B, A_xy );
49      phi_z[1] = angle2( A, B, A_xy );
50
51      double d, delta = 1.e36;
52      int ii = 0, jj = 0, kk = 0;
53      for ( int i = 0; i < 2; i++ ) {
54         for ( int j = 0; j < 2; j++ ) {
55            for ( int k = 0; k < 2 ; k++ ) {
56
57               d = cos( phi_x[i] ) * cos( phi_x[i] ) +
58                   cos( phi_y[j] ) * cos( phi_y[j] ) +
59                   cos( phi_z[k] ) * cos( phi_z[k] );
60               if ( fabs( d - 1. ) < delta ) {
61                  ii = i;
62                  jj = j;
63                  kk = k;
64                  delta = fabs( d - 1. );
65               }
66            }
67         }
68      }
69
70      double th = phi_z[kk];
71      double ph = atan( cos( phi_y[jj] ) / cos( phi_x[ii] ) );
72
73      cout << "Derived Angles (deg): " << endl
74           << "   phi_x: " << phi_x[ii] * va::R2D << endl
75           << "   phi_y: " << phi_y[jj] * va::R2D << endl
76           << "   phi_z: " << phi_z[kk] * va::R2D << endl
77           << "   theta: " << th * va::R2D << endl
78           << "   phi:   " << ph * va::R2D << endl;
79
80      va::Vector u = sin( th ) * cos( ph ) * va::Vector( 1., 0., 0. ) +
81                     sin( th ) * sin( ph ) * va::Vector( 0., 1., 0. ) +
82                     cos( th ) * va::Vector( 0., 0., 1. );
83
84      va::Rotation R( va::Vector( 0., 0., 1. ), u );   // find rotation that takes k to u
85      va::sequence s = factor( R, va::XYZ );
86
87      cout << "pyr = " << +s.first * va::R2D  << "\t" << +s.second * va::R2D << "\t" << s.third * va::R2D << endl;
88      cout << "pyr = " << +s.first * va::R2D  << "\t" << -s.second * va::R2D << "\t" << s.third * va::R2D << endl;
89      cout << "pyr = " << -s.first * va::R2D  << "\t" << +s.second * va::R2D << "\t" << s.third * va::R2D << endl;
90      cout << "pyr = " << -s.first * va::R2D  << "\t" << -s.second * va::R2D << "\t" << s.third * va::R2D << endl;
91
92      cout << "Total effective yaw = " << acos( cos( s.first ) * cos( s.second ) ) * va::R2D << endl;
93
94      return EXIT_SUCCESS;
95   }
```

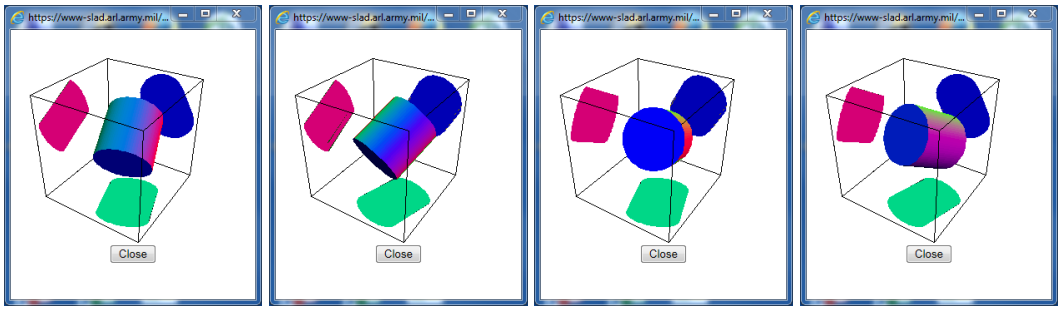

Fig. 2. An oriented cylinder for $\phi_p = \pm 30°$ and $\phi_y = \pm 15°$ is shown with its projected area in the x-y plane in dark blue, the x-z plane in green, and the y-z plane in red. All of these have the same projected areas on each of the orthogonal planes. The actual orientations are not quite the same, but these are superficial differences since the effective yaw angle and the impact angle as defined by FATEPEN are all the same (33.2259° in this example) and thus give the same results for plate impact.

4.2 Cuboid Orientation Sample Case

Consider an RPP with $L = 3$ cm, $W = 2$ cm, and $T = 1$ cm, and let the projected areas be $A_{xy} = 4.5$ cm^2, $A_{xz} = 6.5$ cm^2, $A_{yz} = 3.5$ cm^2. When this information is given to the `orient.cpp` program (Listing 1), it finds 4 different orientations that will work, as displayed in Fig. 3, demonstrating that the orientation is not unique.

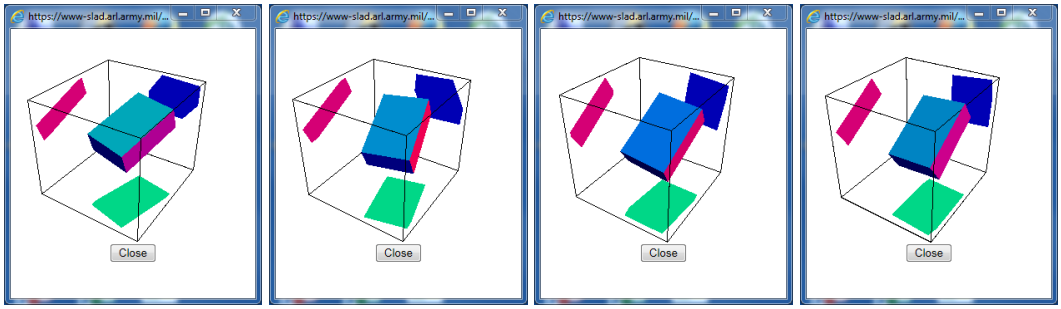

Fig. 3. *The projected area on the orthogonal planes does not uniquely determine the cuboid orientation*. The oriented cuboid ($L = 3$ cm, $W = 2$ cm, $T = 1$ cm) is shown with its projected area in the x-y plane in dark blue (4.5 cm^2), the x-z plane in green (6.5 cm^2), and the y-z plane in red (3.5 cm^2). All 4 have the same projected areas on each of the orthogonal planes, although we see that the actual orientations are not the same. One consequence of this is that the *impact angle*, μ, which is the minimum angle that a face makes with the x-y plane, along with the *effective yaw angle*, ϕ_{eff}, have different values for the 4 orientations—and this will lead to different predictions from FATEPEN.

Next, we want to see what difference, if any, these different orientations will make in FATEPEN. The cuboid penetrator is taken to be steel 4140 with a Brinell hardness

of 300. It has a length of 3 cm = 1.1811 inches, width of 2 cm = 0.7874 inch, and thickness of 1 cm = 0.3937 inch. Using a steel density of 7.83 g/cm^3, it has a mass of 725 gr. The plate target is taken to be 0.25-inch steel with a Brinell hardness of 150. All impacts were at 0° obliquity. The cuboid is initially aligned so that its length is along the z-axis, its width along the x-axis, and it thickness along the y-axis. Then it is given the pitch-yaw-roll rotation sequence listed in Table 2 to give it the orientation shown in Fig. 3, resulting in the following projected areas: $A_{xy} = 4.5$ cm^2, $A_{xz} = 6.5$ cm^2, and $A_{yz} = 3.5$ cm^2. The target is taken to lie in the x-y plane. This table also lists the residual mass m_r, residual velocity v_r, and limit velocity v_L for the 4 orientations. We see that there are significant differences in the residual velocities. It is important to emphasize that all 4 orientations have the same projected areas.

Table 2. FATEPEN predictions for a 725-gr steel cuboid with a striking velocity of 1500 f/s impacting a 0.25-inch mild steel plate using version 3.3.10.3. In each case, the projected areas on the orthogonal planes are the same. Note, in particular, the sensitivity of the residual velocity, v_r.

Pitch (deg)	**Yaw** (deg)	**Roll** (deg)	μ (deg)	ϕ_{eff} (deg)	m_r (gr)	v_r (f/s)	v_L (f/s)
-160.576322	193.931472	1.054946	23.7	23.7	720.341	841.729	1194.12
-166.139458	299.729320	5.443168	61.2	61.2	723.672	547.712	1345.23
176.987194	67.658507	13.759713	67.7	67.7	723.349	376.671	1424.57
205.968278	176.788266	183.822078	26.2	26.2	721.834	830.670	1195.00

4.2.1 Impact Angle and Effective Yaw Angle

Let us also document how the impact angle and effective yaw angles are calculated to check on the values output by FATEPEN. Let us consider the area vector for the front, side, and top of the cuboid. Initially,

$$\boldsymbol{A}_{\text{f}} = WT\,\hat{\mathbf{k}}, \quad \boldsymbol{A}_{\text{s}} = LT\,\hat{\mathbf{i}}, \quad \text{and} \quad \boldsymbol{A}_{\text{t}} = LW\,\hat{\mathbf{j}}. \tag{55}$$

And after the pitch-yaw-roll rotation sequence, these vectors can be calculated by adapting Eqs. 30–32:

$$\boldsymbol{A}'_{\text{f}} = WT[\,\sin\phi_y\hat{\mathbf{i}} - \cos\phi_y \sin\phi_p\hat{\mathbf{j}} + \cos\phi_y\cos\phi_p\hat{\mathbf{k}}], \tag{56}$$

$$\begin{aligned}\boldsymbol{A}'_{\text{s}} = LT[\;&(\cos\phi_r\cos\phi_y)\hat{\mathbf{i}}+\\ &(\sin\phi_r\cos\phi_p + \cos\phi_r\sin\phi_y\sin\phi_p)\hat{\mathbf{j}}+\\ &(\sin\phi_r\sin\phi_p - \cos\phi_r\sin\phi_y\cos\phi_p)\hat{\mathbf{k}}],\end{aligned} \tag{57}$$

$$\begin{aligned}\boldsymbol{A}'_{\text{t}} = LW[&(-\sin\phi_r\cos\phi_y)\hat{\mathbf{i}}+\\ &(\cos\phi_r\cos\phi_p-\sin\phi_r\sin\phi_y\sin\phi_p)\hat{\mathbf{j}}+\\ &(\cos\phi_r\sin\phi_p+\sin\phi_r\sin\phi_y\cos\phi_p)\hat{\mathbf{k}}].\end{aligned} \tag{58}$$

The *effective yaw angle*, ϕ_{eff}, is the angle between the length of the cuboid and the target normal. It is obtained from the dot product between $\boldsymbol{A}'_{\text{f}}$ and $\hat{\mathbf{k}}$, so from Eq. 56, $\phi_{\text{eff}} = \cos^{-1}(\cos\phi_y\cos\phi_p)$. However, if this angle turns out to be greater than 90°, then we should use the back face. Therefore, the correct angle is actually

$$\phi_{\text{eff}} = \min[\cos^{-1}(\cos\phi_y\cos\phi_p),\ \pi-\cos^{-1}(\cos\phi_y\cos\phi_p)]. \tag{59}$$

The *contact angle*, μ, is the angle between each face of the cuboid and the target normal, and the *impact angle*, μ, is the minimum of all the contact angles. Therefore, from Eqs. 55, 56, and 57,

$$\begin{aligned}\mu = \min[&\cos^{-1}(\cos\phi_y\cos\phi_p),\\ &\pi-\cos^{-1}(\cos\phi_y\cos\phi_p),\\ &\cos^{-1}(\sin\phi_r\sin\phi_p-\cos\phi_r\sin\phi_y\cos\phi_p),\\ &\pi-\cos^{-1}(\sin\phi_r\sin\phi_p-\cos\phi_r\sin\phi_y\cos\phi_p),\\ &\cos^{-1}(\cos\phi_r\sin\phi_p+\sin\phi_r\sin\phi_y\cos\phi_p),\\ &\pi-\cos^{-1}(\cos\phi_r\sin\phi_p+\sin\phi_r\sin\phi_y\cos\phi_p)].\end{aligned} \tag{60}$$

Applying these formulas for the 4 orientations shown in Table 2, we find $\mu = 23.7454^\circ$, $\mu = 30.4564^\circ$, $\mu = 24.5429^\circ$, and $\mu = 26.1524^\circ$. Notice that 2 of these angles disagree with the values output by FATEPEN as listed in Table 2. Even though the FATEPEN documentation states that μ is the minimum contact angle, it seems to always equal the effective yaw angle instead.

A newer version of FATEPEN (3.3.18.0) was also run,* which produced the results listed in Table 3.

*I thank Timothy Mallory for making runs with both version 3.3.16.10 and 3.3.18.0, which produced the same results shown in Table 3.

Table 3. FATEPEN predictions for a 725-gr steel cuboid with a striking velocity of 1500 f/s impacting a 0.25-inch mild steel plate using version 3.3.18.0. In each case, the projected areas on the orthogonal planes are the same.

Pitch (deg)	**Yaw** (deg)	**Roll** (deg)	μ (deg)	ϕ_{eff} (deg)	m_r (gr)	v_r (f/s)	v_L (f/s)
-160.576322	193.931472	1.054946	23.7	23.7	720.338	757.72	1281.89
-166.139458	299.729320	5.443168	61.2	61.2	723.671	317.05	1456.00
176.987194	67.658507	13.759713	0.0	67.7	...	0.0	1549.32
205.968278	176.788266	183.822078	26.2	26.2	721.833	747.18	1282.89

So version 3.3.18.0 seems to say that the residual velocity can be half the striking velocity, roughly one-fourth the striking velocity, or zero, depending upon angles that we have no way of measuring during the test, even though the presented areas on each of the orthogonal planes are all the same.

5. Conclusions

What are the implications from these results? In many cases, all we have are yaw card estimates of the actual impact presented area, so that at best we only have the projected area on one plane, the x-y plane. This may be adequate for a cylinder, but it will not determine the orientation of a cuboid. What we have shown here is that even if we have ideal measurements of the projected area on orthogonal planes, this is still not enough to resolve the orientation. As far as FATEPEN is concerned, the orientation is specified once the impact angle μ and the yaw angle ϕ_{eff} are known. I am not aware of any measurement that can determine these during the tests. Thus, I think we are justified in stating that this analysis shows that cuboids are not good candidates for FATEPEN validation because the results are highly dependent upon angles that we have no known way of measuring and no way of uniquely computing from the measured projected areas.

What are the implications for trying to model natural fragments in FATEPEN? If we expect to model natural fragments as multifaceted solids, then it seems we have a real problem since we have no way of knowing the many contact angles, let alone the impact angle. On the other hand, if we model natural fragments as the simplest shape that has the same measured presented area, then yawed cylinders offer more promise and can be checked against experiment.

Part of the problem here is that the FATEPEN code turns all shapes into “equivalent

cylinders".* So while we may be able to compute the contact angle of each face of the cuboid, that is not necessarily the shape that the FATEPEN code is dealing with at impact. We are much better off using cylinders since we can be assured that FATEPEN is using the same shape. Further, it is more robust since we only need to control the impact presented area and there is only one contact angle, which is the angle the RCC face makes with the target normal. Once we know the impact presented area and the cylinder dimensions, the impact angle is uniquely determined. See Appendix B for an implementation of this procedure.

*See Yatteau et al.,[5] Section 2.1.4 on p. 2-5 and Section 4.1 on p. 4-1.

6. References

1. Yatteau JD, Zernow RH, Recht GW, Edquist KT. FATEPEN. Version 3.0.0b. Terminal Ballistic Penetration Model. Applied Research Associates (ARA) Project 4714, prepared for Naval Surface Warfare Center, Dahlgren, VA; 1999 Jan.

2. Project THOR. The resistance of various metallic materials to perforation by steel fragments: empirical relationships for fragment residual velocity and residual weight. Aberdeen Proving Ground (MD): Army Ballistic Research Laboratory (US); 1961 Apr. Report No.: TR-47.

3. Kuipers JB. Quaternions and rotation sequences: a primer with applications to orbits, aerospace, and virtual reality. Princeton (NJ): Princeton University Press; 2002.

4. Mallory TD. FATEPEN steel-on-steel V50 estimates. Aberdeen Proving Ground (MD): Army Research Laboratory (US); unpublished paper, 2015 Nov.

5. Yatteau JD, Zernow RH, Recht GW, Edquist KT. FATEPEN fast air target encounter penetration (Ver. 3.0.0) terminal ballistic penetration model. Littleton (CO): Applied Research Associates, Inc.; 2001 Sep., revised 2005 Feb 2. (Analyst's manual; vol. 1).

INTENTIONALLY LEFT BLANK.

Appendix A. Formula for an Euler Sequence of Rotations

Let the rotation be a Euler sequence about 3 (distinct or repeated) principal axes unit-vectors $\hat{\mathbf{e}}_1$, $\hat{\mathbf{e}}_2$, $\hat{\mathbf{e}}_3$ as follows:

- First, a rotation of ϕ_1 about $\hat{\mathbf{e}}_1$:

$$R_1 = R_{\hat{\mathbf{e}}_1}(\phi_1). \tag{A-1}$$

- Second, a rotation of ϕ_2 about the transformed $\hat{\mathbf{e}}_2$ unit vector:

$$R_2 = R_{\hat{\mathbf{e}}'_2}(\phi_2), \tag{A-2}$$

  where $\hat{\mathbf{e}}'_2 = R_1\hat{\mathbf{e}}_2$.

- Third, a rotation of ϕ_3 about the (doubly) transformed $\hat{\mathbf{e}}_3$ unit vector:

$$R_3 = R_{\hat{\mathbf{e}}''_3}(\phi_3), \tag{A-3}$$

  where $\hat{\mathbf{e}}''_3 = R_2\hat{\mathbf{e}}'_3 = R_2R_1\hat{\mathbf{e}}_3$.

The total combined rotation is

$$R = R_3R_2R_1 = R_{\hat{\mathbf{e}}''_3}(\phi_3)R_{\hat{\mathbf{e}}'_2}(\phi_2)R_{\hat{\mathbf{e}}_1}(\phi_1), \tag{A-4}$$

where the rotations are applied successively from right to left.

Now, the rotation about $\hat{\mathbf{e}}'_2$ can be obtained by undoing the effect of the first rotation, performing the rotation about $\hat{\mathbf{e}}_2$, and then rotating back:

$$\begin{aligned} R_2 &= R_{\hat{\mathbf{e}}'_2}(\phi_2) \\ &= R_1R_{\hat{\mathbf{e}}_2}(\phi_2)R_1^{-1} \\ &= R_{\hat{\mathbf{e}}_1}(\phi_1)R_{\hat{\mathbf{e}}_2}(\phi_2)R^{-1}_{\hat{\mathbf{e}}_1}(\phi_1). \end{aligned} \tag{A-5}$$

Similarly,

$$
\begin{aligned}
R_3 &= R_{\hat{\mathbf{e}}_3''}(\phi_3) \\
&= R_2 R_{\hat{\mathbf{e}}_3'}(\phi_3) R_2^{-1} \\
&= R_2 R_1 R_{\hat{\mathbf{e}}_3}(\phi_3) R_1^{-1} R_2^{-1} \quad \text{(and using Eq. A-5)} \\
&= [R_{\hat{\mathbf{e}}_1}(\phi_1) R_{\hat{\mathbf{e}}_2}(\phi_2) R_{\hat{\mathbf{e}}_1}^{-1}(\phi_1)] R_{\hat{\mathbf{e}}_1}(\phi_1) R_{\hat{\mathbf{e}}_3}(\phi_3) R_{\hat{\mathbf{e}}_1}^{-1}(\phi_1) [R_{\hat{\mathbf{e}}_1}(\phi_1) R_{\hat{\mathbf{e}}_2}(\phi_2) R_{\hat{\mathbf{e}}_1}^{-1}(\phi_1)]^{-1} \\
&= R_{\hat{\mathbf{e}}_1}(\phi_1) R_{\hat{\mathbf{e}}_2}(\phi_2) R_{\hat{\mathbf{e}}_1}^{-1}(\phi_1) R_{\hat{\mathbf{e}}_1}(\phi_1) R_{\hat{\mathbf{e}}_3}(\phi_3) R_{\hat{\mathbf{e}}_1}^{-1}(\phi_1) R_{\hat{\mathbf{e}}_1}(\phi_1) R_{\hat{\mathbf{e}}_2}^{-1}(\phi_2) R_{\hat{\mathbf{e}}_1}^{-1}(\phi_1) \\
&= R_{\hat{\mathbf{e}}_1}(\phi_1) R_{\hat{\mathbf{e}}_2}(\phi_2) R_{\hat{\mathbf{e}}_3}(\phi_3) R_{\hat{\mathbf{e}}_2}^{-1}(\phi_2) R_{\hat{\mathbf{e}}_1}^{-1}(\phi_1). \qquad \text{(A-6)}
\end{aligned}
$$

The combined rotation collapses into something very simple:

$$
\begin{aligned}
R &= R_3 R_2 R_1 \\
&= R_{\hat{\mathbf{e}}_1}(\phi_1) R_{\hat{\mathbf{e}}_2}(\phi_2) R_{\hat{\mathbf{e}}_3}(\phi_3) R_{\hat{\mathbf{e}}_2}^{-1}(\phi_2) R_{\hat{\mathbf{e}}_1}^{-1}(\phi_1) R_{\hat{\mathbf{e}}_1}(\phi_1) R_{\hat{\mathbf{e}}_2}(\phi_2) R_{\hat{\mathbf{e}}_1}^{-1}(\phi_1) R_{\hat{\mathbf{e}}_1}(\phi_1) \\
&= R_{\hat{\mathbf{e}}_1}(\phi_1) R_{\hat{\mathbf{e}}_2}(\phi_2) R_{\hat{\mathbf{e}}_3}(\phi_3). \qquad \text{(A-7)}
\end{aligned}
$$

Thus, we get the result that *the combined rotation is equal to the successive rotations about the* ***original*** *unit vectors but applied in* ***reverse*** *order*. Explicitly, this means

$$
\boxed{R_{\hat{\mathbf{e}}_3''}(\phi_3) R_{\hat{\mathbf{e}}_2'}(\phi_2) R_{\hat{\mathbf{e}}_1}(\phi_1) = R_{\hat{\mathbf{e}}_1}(\phi_1) R_{\hat{\mathbf{e}}_2}(\phi_2) R_{\hat{\mathbf{e}}_3}(\phi_3)} \quad . \qquad \text{(A-8)}
$$

What we have shown so far is more or less a plausibility argument. Here is a derivation using quaternions. We use the notation

$$
q_{\hat{\mathbf{u}}}(\phi) = \cos\frac{\phi}{2} + \hat{\mathbf{u}} \sin\frac{\phi}{2} \qquad \text{(A-9)}
$$

for the unit quaternion that represents a counterclockwise rotation of ϕ radians about the unit vector $\hat{\mathbf{u}}$. Then, referring to Eqs. A-1, A-2, and A-3,

$$
R_1 = q_{\hat{\mathbf{e}}_1}(\phi_1). \qquad \text{(A-10)}
$$

$$
R_2 = q_{\hat{\mathbf{e}}_2'}(\phi_2) = \cos\frac{\phi_2}{2} + \hat{\mathbf{e}}_2' \sin\frac{\phi_2}{2}, \qquad \text{(A-11)}
$$

where

$$
\hat{\mathbf{e}}_2' = q_{\hat{\mathbf{e}}_1}(\phi_1) \hat{\mathbf{e}}_2 q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1), \qquad \text{(A-12)}
$$

so that

$$\begin{aligned} q_{\hat{\mathbf{e}}_2'}(\phi_2) &= \cos\frac{\phi_2}{2} + \hat{\mathbf{e}}_2' \sin\frac{\phi_2}{2} \\ &= \cos\frac{\phi_2}{2} + q_{\hat{\mathbf{e}}_1}(\phi_1)\hat{\mathbf{e}}_2 q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1)\sin\frac{\phi_2}{2} \\ &= q_{\hat{\mathbf{e}}_1}(\phi_1)\left(\cos\frac{\phi_2}{2} + \hat{\mathbf{e}}_2\sin\frac{\phi_2}{2}\right) q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1) \\ &= q_{\hat{\mathbf{e}}_1}(\phi_1) q_{\hat{\mathbf{e}}_2}(\phi_2) q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1), \end{aligned} \tag{A-13}$$

and

$$R_3 = q_{\hat{\mathbf{e}}_3''}(\phi_3) = \cos\frac{\phi_3}{2} + \hat{\mathbf{e}}_3''\sin\frac{\phi_3}{2}, \tag{A-14}$$

where

$$\begin{aligned} \hat{\mathbf{e}}_3'' &= q_{\hat{\mathbf{e}}_2'}(\phi_2)\hat{\mathbf{e}}_3' q_{\hat{\mathbf{e}}_2'}^{-1}(\phi_2) \\ &= q_{\hat{\mathbf{e}}_2'}(\phi_2) q_{\hat{\mathbf{e}}_1}(\phi_1)\hat{\mathbf{e}}_3 q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1) q_{\hat{\mathbf{e}}_2'}^{-1}(\phi_2) \\ &= [q_{\hat{\mathbf{e}}_1}(\phi_1) q_{\hat{\mathbf{e}}_2}(\phi_2) q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1)] q_{\hat{\mathbf{e}}_1}(\phi_1)\hat{\mathbf{e}}_3 q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1)[q_{\hat{\mathbf{e}}_1}(\phi_1) q_{\hat{\mathbf{e}}_2}(\phi_2) q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1)]^{-1} \\ &= q_{\hat{\mathbf{e}}_1}(\phi_1) q_{\hat{\mathbf{e}}_2}(\phi_2) q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1) q_{\hat{\mathbf{e}}_1}(\phi_1)\hat{\mathbf{e}}_3 q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1) q_{\hat{\mathbf{e}}_1}(\phi_1) q_{\hat{\mathbf{e}}_2}^{-1}(\phi_2) q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1) \\ &= q_{\hat{\mathbf{e}}_1}(\phi_1) q_{\hat{\mathbf{e}}_2}(\phi_2)\hat{\mathbf{e}}_3 q_{\hat{\mathbf{e}}_2}^{-1}(\phi_2) q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1), \end{aligned} \tag{A-15}$$

so that

$$\begin{aligned} q_{\hat{\mathbf{e}}_3''}(\phi_3) &= \cos\frac{\phi_3}{2} + \hat{\mathbf{e}}_3''\sin\frac{\phi_3}{2} \\ &= \cos\frac{\phi_3}{2} + q_{\hat{\mathbf{e}}_1}(\phi_1) q_{\hat{\mathbf{e}}_2}(\phi_2)\hat{\mathbf{e}}_3 q_{\hat{\mathbf{e}}_2}^{-1}(\phi_2) q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1)\sin\frac{\phi_3}{2} \\ &= q_{\hat{\mathbf{e}}_1}(\phi_1) q_{\hat{\mathbf{e}}_2}(\phi_2)\left(\cos\frac{\phi_3}{2} + \hat{\mathbf{e}}_3\sin\frac{\phi_3}{2}\right) q_{\hat{\mathbf{e}}_2}^{-1}(\phi_2) q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1) \\ &= q_{\hat{\mathbf{e}}_1}(\phi_1) q_{\hat{\mathbf{e}}_2}(\phi_2) q_{\hat{\mathbf{e}}_3}(\phi_3) q_{\hat{\mathbf{e}}_2}^{-1}(\phi_2) q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1). \end{aligned} \tag{A-16}$$

Therefore, the total combined rotation is

$$\begin{aligned} R = R_3 R_2 R_1 &= q_{\hat{\mathbf{e}}_3''}(\phi_3) q_{\hat{\mathbf{e}}_2'}(\phi_2) q_{\hat{\mathbf{e}}_1}(\phi_1) \\ &= [q_{\hat{\mathbf{e}}_1}(\phi_1) q_{\hat{\mathbf{e}}_2}(\phi_2) q_{\hat{\mathbf{e}}_3}(\phi_3) q_{\hat{\mathbf{e}}_2}^{-1}(\phi_2) q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1)][q_{\hat{\mathbf{e}}_1}(\phi_1) q_{\hat{\mathbf{e}}_2}(\phi_2) q_{\hat{\mathbf{e}}_1}^{-1}(\phi_1)] q_{\hat{\mathbf{e}}_1}(\phi_1) \\ &= q_{\hat{\mathbf{e}}_1}(\phi_1) q_{\hat{\mathbf{e}}_2}(\phi_2) q_{\hat{\mathbf{e}}_3}(\phi_3), \end{aligned} \tag{A-17}$$

as was to be shown.

The program in Listing A-1 provides a way of checking this result.

Listing A-1. reverse.cpp

```
// reverse.cpp: program to check that a pitch-yaw-roll rotation sequence
// about transformed axes is equivalent to the reverse sequence about fixed axes

#include "Rotation.h"
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <iomanip>
using namespace std;

int main( int argc, char* argv[] ) {

   va::Vector i( 1., 0., 0. ), j( 0., 1., 0. ), k( 0., 0., 1. );   // three unit vectors
   va::Vector i1, j1, k1, i2, j2, k2, i3, j3, k3;                   // transformed unit vectors
   va::Rotation Rp, Ry, Rr;                                         // rotations for pitch, yaw and roll
   double p = 0., y = 0., r = 0.;
   if ( argc == 4 ) {

      p = atof( argv[1] ) * va::D2R;
      y = atof( argv[2] ) * va::D2R;
      r = atof( argv[3] ) * va::D2R;
   }

   // first, perform pitch about x-axis
   Rp = va::Rotation( i, p );
   i1 = Rp * i;
   j1 = Rp * j;
   k1 = Rp * k;

   // second, perform yaw about transformed y-axis
   Ry = va::Rotation( j1, y );
   i2 = Ry * i1;
   j2 = Ry * j1;
   k2 = Ry * k1;

   // third, perform roll about doubly transformed z-axis
   Rr = va::Rotation( k2, r );
   i3 = Rr * i2;
   j3 = Rr * j2;
   k3 = Rr * k2;

   cout << "First done the conventional way:" << endl;
   cout << setprecision(6) << fixed;
   cout << "i3 = " << i3 << endl;
   cout << "j3 = " << j3 << endl;
   cout << "k3 = " << k3 << endl << endl;

   // perform rotation sequence in reverse order about fixed axes
   va::Rotation R = va::Rotation( i, p ) * va::Rotation( j, y ) * va::Rotation( k, r );

   cout << "Now, the same rotations done in reverse order about fixed axes:" << endl;
   cout << "i  = " << R * i << endl;
   cout << "j  = " << R * j << endl;
   cout << "j  = " << R * k << endl << endl;

   va::Vector v = 3.4 * i - 5.7 * j + 2.3 * k;   // an arbitrary vector

   cout << "original vector:" << endl;
   cout << v << endl << endl;

   cout << "transformed vector using the conventional procedure:" << endl;
   cout << Rr * Ry * Rp * v << endl << endl;

   cout << "transformed vector using the reversed order about fixed axes:" << endl;
   cout << R * v << endl;

   return 0;
}
```

For example, the commands

```
c++ -O2 -Wall -std=c++11 -o reverse reverse.cpp -lm
./reverse  65. -137. 54.
```

will print the results

```
First done the conventional way:
i3 = -0.429879 -0.021405 0.902633
j3 = 0.591678 0.748463 0.299535
k3 = -0.681998 0.662832 -0.309083

Now, the same rotations done in reverse order about fixed axes:
i  = -0.429879 -0.021405 0.902633
j  = 0.591678 0.748463 0.299535
j  = -0.681998 0.662832 -0.309083

original vector:
3.400000 -5.700000 2.300000

transformed vector using the conventional procedure:
-6.402747 -2.814501 0.650707

transformed vector using the reversed order about fixed axes:
-6.402747 -2.814501 0.650707
```

Appendix B. Yaw Angle of a Cylinder as a Function of Shape Factor

The *dimensionless shape factor* γ is defined by the equation

$$A_p = \gamma V^{2/3}, \tag{B-1}$$

where A_p is the projected area and V is the volume. The formula for the dimensionless shape factor of a right-circular cylinder (RCC) as a function of yaw angle ϕ_y is

$$\gamma(\phi_y) = a \sin\phi_y + b \cos\phi_y, \tag{B-2}$$

where

$$a \equiv \left(\frac{\pi}{4}\frac{L}{D}\right)^{-2/3} \frac{L}{D} \quad \text{and} \quad b \equiv \left(\frac{\pi}{4}\frac{L}{D}\right)^{-2/3} \frac{\pi}{4}. \tag{B-3}$$

The yaw angle that gives the maximum shape factor is obtained by setting the derivative with respect to ϕ_y equal to zero and solving for ϕ_y:

$$\left(\frac{d\gamma}{d\phi_y}\right)_{\gamma=\gamma_{\max}} = a\cos\phi_y - b\sin\phi_y = 0, \tag{B-4}$$

which gives

$$\phi_{y\,\gamma=\gamma_{\max}} = \tan^{-1}\left(\frac{a}{b}\right). \tag{B-5}$$

To simplify the notation, let $\widehat{\phi_y}$ denote this angle: $\widehat{\phi_y} \equiv \phi_{y\,\gamma=\gamma_{\max}}$. Then,

$$a = \gamma_{\max}\sin\widehat{\phi_y} \quad \text{and} \quad b = \gamma_{\max}\cos\widehat{\phi_y}, \tag{B-6}$$

so that Eq. B-2 can be written as

$$\gamma(\phi_y) = \begin{cases} \gamma_{\max}\cos(\phi_y - \widehat{\phi_y}) & \text{if } \phi_y > \widehat{\phi_y} \\ \gamma_{\max}\cos(\widehat{\phi_y} - \phi_y) & \text{if } \phi_y < \widehat{\phi_y} \end{cases}, \tag{B-7}$$

where $\widehat{\phi_y} = \cos^{-1}(b/\gamma_{\max})$. Solving for the yaw angle, we get

$$\boxed{\phi_y = \begin{cases} \cos^{-1}(b/\gamma_{\max}) + \cos^{-1}(\gamma/\gamma_{\max}) & \text{if } \gamma < b \\ \cos^{-1}(b/\gamma_{\max}) - \cos^{-1}(\gamma/\gamma_{\max}) & \text{if } \gamma \geq b \end{cases}}, \tag{B-8}$$

which shows that we can easily get the orientation (yaw angle) of an RCC from just the impact presented area.

The code in Listing B-1 implements this procedure by sampling dimensionless shape factors from a lognormal distribution to generate the appropriate yawed RCC. The program may be compiled and run with the following commands:

```
c++ -O2 -Wall -std=c++11 -o cyl cyl.cpp -lm
./cyl
```

Listing B-1. cyl.cpp

```
// cyl.cpp: Implementation of an algorithm for generating FATEPEN RCCs to represent a specified shape factor.
//          Given a dimensionless shape factor, generates the L/D and yaw angle for the RCC to represent it.
//          The RCCs are disk-like in that the L/D <= Pi/4, which is necessary since [0.5,4.5]
//          is the range of shape factors for artillery and rod-like RCCs don't span this range.
// R. Saucier, October 2011

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cmath>
#include <random>
#include <chrono>
using namespace std;

int main( int argc, char* argv[] ) {

   const int    N      = 1000;          // number of samples
   const double R2D    = 180. / M_PI;   // to convert from radians to degrees
   const double SF_MIN = 0.5;           // minimum shape factor (found from artillery fragments)
   const double SF_MAX = 4.5;           // maximum shape factor (found from artillery fragments)

   // default values for the shape factor lognormal distribution from 122mm, 152mm and 155mm artillery
   double mu    = 0.590494;   // these two parameters characterize the lognormal shape factor distribution
   double sigma = 0.323433;   // with mode = 1.63, median = 1.80 and mean = 1.90

   // ability to override the default values from the command line by specifying min and max shape factor
   // that is meant to capture 95% of the complete distribution (from 0.025 to 0.975)
   if ( argc == 3 ) {

      double sfmin = atof( argv[1] );
      double sfmax = atof( argv[2] );
      mu    = 0.5 * log( sfmin * sfmax );
      sigma = log( sfmax / sfmin ) / ( 2. * M_SQRT2 * 1.3859 );
   }

   unsigned int seed = std::chrono::high_resolution_clock::now().time_since_epoch().count();
   std::mt19937 rng( seed );                                        // Mersenne Twister engine
   std::lognormal_distribution<double>    lognormal( mu, sigma );     // lognormal shape factor distribution
   std::uniform_real_distribution<double> uniform( 0.069, M_PI_4 );   // uniform distribution for L/D

   double a, b, c, th, sf_min, sf_max, sf, l_d, l, d, yaw, V = 1.;   // here we use a fragment with unit volume
   std::cout << std::setprecision(6) <<std::fixed;

   for ( int n = 0; n < N; n++ ) {

      // normally, the shape factor would be provided, but here we get a shape factor within bounds [SF_MIN, SF_MAX]
      do { sf = lognormal( rng ); } while ( sf < SF_MIN || sf > SF_MAX );

      // now we want to realize this shape factor with a cylinder
      do {
         l_d    = uniform( rng );
         c      = pow( M_PI_4 * l_d, -2./3. );
         a      = c * l_d;
         b      = c * M_PI_4;
         sf_min = a;
         sf_max = sqrt( a * a + b * b );
      } while ( sf < sf_min || sf > sf_max );

      if ( sf < b ) th = acos( b / sf_max ) + acos( sf / sf_max );
      else          th = acos( b / sf_max ) - acos( sf / sf_max );

      d   = pow( V / ( M_PI_4 * l_d ), 1./3. );
      l   = d * l_d;
      yaw = th * R2D;

      std::cout << l_d << "\t" << yaw << std::endl;
   }

   return EXIT_SUCCESS;
}
```

INTENTIONALLY LEFT BLANK.

List of Symbols, Abbreviations, and Acronyms

TERMS:

FATEPEN: Fast Air Target Encounter Penetration

RCC: right-circular cylinder

RPP: rectangular parallelepiped

MATHEMATICAL SYMBOLS:

$R_{\hat{\mathbf{e}}}(\theta)$: Rotation about the unit vector $\hat{\mathbf{e}}$ through the angle θ

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT

1 (PDF) GOVT PRINTG OFC A MALHOTRA

1 (HC) NVL SURFC WARFARE CTR ATTN: D DICKINSON G24 6138 NORC AVE STE 313 DAHLGREN VA 22448-5157

1 (HC) APPLIED RESEARCH ASSOCIATES ATTN: R ZERNOW 10720 BRADFORD RD STE 110 LITTLETON CO 80127-4298

ABERDEEN PROVING GROUND

8 (PDF) RDRL SLB D
J COLLINS
RDRL SLB G
D CARABETTA
T MALLORY
RDRL SLB S
J AUTEN
R DIBELKA
S MORRISON
N REED
R SAUCIER

1 (HC) R SAUCIER